Foliage Discrimination using a Rotating Ladar

Andres Castano and Larry Matthies

Jet Propulsion Laboratory
4800 Oak Grove Dr.
Pasadena, CA 91109
{andres/lhm}@telerobotics.jpl.nasa.gov

Abstract

An outdoor environment presents to a robot objects that are drivable, like tall grass and small bushes, or non-drivable, like trees and rocks. Traditionally, because of the difficulty of discriminating between these classes, a robot searches for paths free of objects of either class. Although this approach prevents collisions with objects missclassified as drivable, it also eliminates a large number of non-free drivable paths and by doing so, it might eliminate the only path to a desired destination. In this paper we present of a real time algorithm that detects foliage using range estimates from a rotating laser. Objects not classified as foliage are conservatively labeled as non-drivable obstacles. In contrast to related work that uses range statistics to classify the objects, we exploit the expected localities of an obstacle, in both space and time. The Urbie robot is presently using this algorithm to discriminate drivable grass from obstacles during outdoors autonomous navigation tasks.

1 Introduction

A major problem for autonomous cross-country navigation of robots is the discrimination between drivable and non-drivable objects in the path. Until a few years ago, outdoors robotics was mostly avoided, in part because many core research problems (e.g., path planning, mapping, etc.) could be studied indoors, decoupling them from the uncertainty associated with unstructured environments. The main exception was outdoors navigation of man-made roads, as in the case of the Navlab vehicle and its successors [1]. As the state of all-terrain sensors improves, the development of cross-country navigation platforms, like military surveillance robots and search and rescue vehicles, is becoming cost-effective.

The laser radar (ladar) is one of the sensors that is becoming cost-effective in cross-country navigation. The ladar is an active sensor that fires a laser beam and then senses its reflection, or return, from the scene. From this return, the sensor estimates the distance, or range, to the object of the scene hit by the beam. The use of a rotating mirror allows the sensor to sweep the scene about an axis, obtaining a 1-D range signal (e.g., [2]); placing a 1-D laser on a tilt unit allows the sensor to sweep an area of the scene, producing a range image (e.g., [3], [4])). Ladars are useful because they provide a range estimate in many situations where it cannot be estimated with a stereo pair or by other means, e.g., night operation, low-frequency high-contrast scenes with shadows, etc.

In this paper we describe a real time algorithm that uses the range estimates of a rotating ladar to detect foliage in an outdoors scenario. This classification allows the inclusion of tall grass to any free path that the robot may select while still allowing it to avoid partially hidden obstacles. Recent approaches to use a ladar to find obstacles partially hidden by grass rely on the statistics of the signal (e.g., [5]). In contrast to these approaches, we exploit the spatial and temporal localities of the objects in the scene to classify the returns. The result is a robust algorithm with a low false alarm rate.

This paper is organized as follows. In Secs. 2 and 3 we describe the ladar and provide an overview of the algorithm. In Secs. 4 and 5 we show how to select non-foliage returns and how to prune them, respectively. In Sec. 6 we describe how we keep track of the state of the scene. In Sec. 7 we describe experimental results using Urbie, an all-terrain autonomous robot. Finally, in Sec. 8 we present our conclusions.

2 The Ladar System

Consider a mobile robot moving in a field of grass, as shown in Fig. 1. This figure shows a top view of a cross-section of the scene, where blades of grass and tree trunks are represented by small and large white circles, respectively. Tree trunks A and B represent the cases where an obstacle is partially hidden by grass and in the clear, respectively. The robot scans the scene parallel to the ground, once every τ sec-

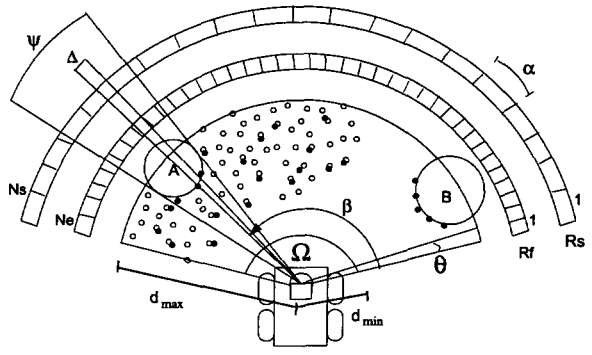

***Figure 1:** Robot using a ladar to scan a scene*

onds. Each scan spans an angle of Ω radians and provides range estimates to objects located farther than d_{min} and closer than d_{max}. Thus, a sampling interval (i.e., the angle between successive fires of the laser) of θ radians yields at most

$$N_e = \Omega/\theta$$

range estimates per scan. Some estimates might be missing, e.g., there might not be an object in the path of the laser within the valid distances or the target might absorb the light at the wavelength of the laser. In Fig. 1, the locations of the scene hit by a beam of the scan are marked with a black circle.

The accuracy of each range estimate is affected by parameters as varied as distance, environmental temperature and color and pose of the target [2]. In practice, within a narrow range interval, the error δ_{d_i} of an estimate can be considered to vary linearly with the distance d_i. Thus, the range estimate of an object located at a distance d_i is

$$r_i = d_i \pm (a\ d_i + b)$$

with

$$a = \frac{\delta_{max} - \delta_{min}}{d_{max} - d_{min}} \quad \text{and} \quad b = \delta_{min} - a\ d_{min}, \quad (1)$$

for $d_{min} \leq d_i \leq d_{max}$.

Finally, we consider negligible the laser divergence within the valid range. Thus, the beam is so narrow that it cannot hit two different objects located at different distances which creates erroneous range estimates formed by a combination of both distances.

3 Algorithm Overview

The goal of the algorithm is to discriminate foliage from other elements of the scene by classifying each return of each scan as FOLIAGE or NOTFOLIAGE. Initially the robot is located in free space, either in a clear or in the middle of a field of grass. After each scan is processed, the algorithm produces three outputs that describe the results at different levels of detail. The most detailed output is the array of final range estimates, R_f, which contains N_e elements, each one corresponding to one return of the scan, as shown in Fig. 1. If we are not interested in the classification of each return then the array R_s summarizes the results that span a given angle α. Hence, R_s has N_s elements where

$$N_s = \Omega/\alpha.$$

Finally, the most general output is the flag *alarm* which is set to indicate that there is an object in the scene that is not foliage.

The pseudo-code of the algorithm is:

```
ANALYZE–LADAR-SCANS ()
1   R_f(1 : N_e) ← 0
2   R_s(1 : N_s) ← 0
3   forever
4       [A, R] ←GET-SCAN-DATA ()
5       [alarm, R_s, R_f] ←CLASSIFY (A, R)
```

After initializing the arrays R_f and R_s we proceed to process the scans. Line 4 reads the scan into two arrays of N_e elements, A and R. The element $A(i)$ is the angle at which the i-th beam was fired while $R(i)$ is the raw range estimate of the object hit by the beam. In the rest of this paper we assume that we have access to N_e, N_s, d_{min} and d_{max}.

In the routine CLASSIFY(), we exploit three locality principles to find NOTFOLIAGE returns. To illustrate these principles we assume that at time t we found an obstacle an angle β, e.g., the tree trunk A in Fig. 1. First, the locality in time of the obstacle indicates that it will be located at around β at time $t + \tau$. Second, we use the locality in space of the obstacle, i.e., an obstacle must have a large size, spaning over a large angle ψ. Thus, if a beam hits the obstacle at angle β then all beams that might hit the obstacles must lie within $\beta \pm \psi$. Finally, we use the locality in space of the clear in the foliage that allowed the laser to hit the partially hidden obstacle, i.e., if a beam at angle β penetrates the foliage, then all its inmediate neighbors fired at $\beta \pm \Delta$, for $\Delta \ll \psi$, are likely to penetrate the foliage through this same clear too. These locality principles hold for any combination of motions of the robot and the obstacles, as long as the sampling interval, θ, and the time between consecutive scans, τ, are sufficiently small.

The pseudo-code of the CLASSIFY routine is

```
CLASSIFY (A, R)
1   [v, nad2v] ←FIND-LOW-FREQ (A, R)
```

2 $R_c \leftarrow$GET-OBST-CANDIDATES $(A, R, v, nad2v)$
3 $R_c \leftarrow$RELAX-OBST (A, R, R_c)
4 $R_f \leftarrow$REMOVE-NEW-OBST (A, R_c)
5 $R_f \leftarrow$REMOVE-THIN-OBST (R_f, W_s, O_s)
6 $R_f \leftarrow$CROP-RANGE (R_f, d_{min}, d_{max})
7 $[alarm, R_s] \leftarrow$UPDATE-AND-ARCHIVE (A, R_c)
8 **return** $(alarm, R_s, R_f)$

The first three routines determine R_c, an array of returns likely to have hit obstacles. The second three routines determine R_f, that classifies each return of the scan. Finally, the routine UPDATE-AND-ARCHIVE saves the results for evalation of future scans. We now discuss these routines.

4 Candidate selection

The first three routines of CLASSIFY select returns that are likely to belong to obstacles.

4.1 Finding low-frequency scan regions

The routine FIND-LOW-FREQ uses the estimates R to find an array R_c where it is easy to identify returns that belong to obstacles. Consider the returns in Fig. 1. If we plot R, as shown in Fig. 2.a, we notice that the problem of locating the obstacles amid the foliage is similar to that of recovering a signal buried in noise. Thus, continuining with the analogy, the obstacles can be recovered better if we filter out some of the foliage returns, increasing the signal-to-noise ratio of the scan.

To select a filter that removes mostly foliage returns, consider all the possible returns within an angle α (i.e., the size of our filter) that are centered around an angle β, as shown in Fig. 1. If the obstacle spans an angle larger than α, any beam fired at $\beta \pm \alpha/2$ can travel, at most, as far as the obstacle. Thus, the largest return of the set is likely to belong to the obstacle if any of the beams hit the obstacle. Hence, in our case, we can use a maximum-value filter that will remove mainly foliage returns. The size of the filter, α, must be larger than the sampling interval θ and smaller than the angle spanned by the smallest object that we want to detect when located at a distance d_{min} from the sensor. The result of applying such filter to nonoverlapping windows of R is shown in Fig. 2.b. Within each window, a black dot indicates the maximum return selected by the filter, a white dot indicates a return filtered out, and a solid line indicates the value of the resulting filtered scan v. Since, in this example, $\alpha \approx 2\ \theta$ then v has $N_s \approx N_e/2$ elements.

The next step is to identify low frequency regions of the array v that might indicate the presence of an obstacle using the magnitude of its first or second derivatives. In Figs. 2.c-d we show approxima-

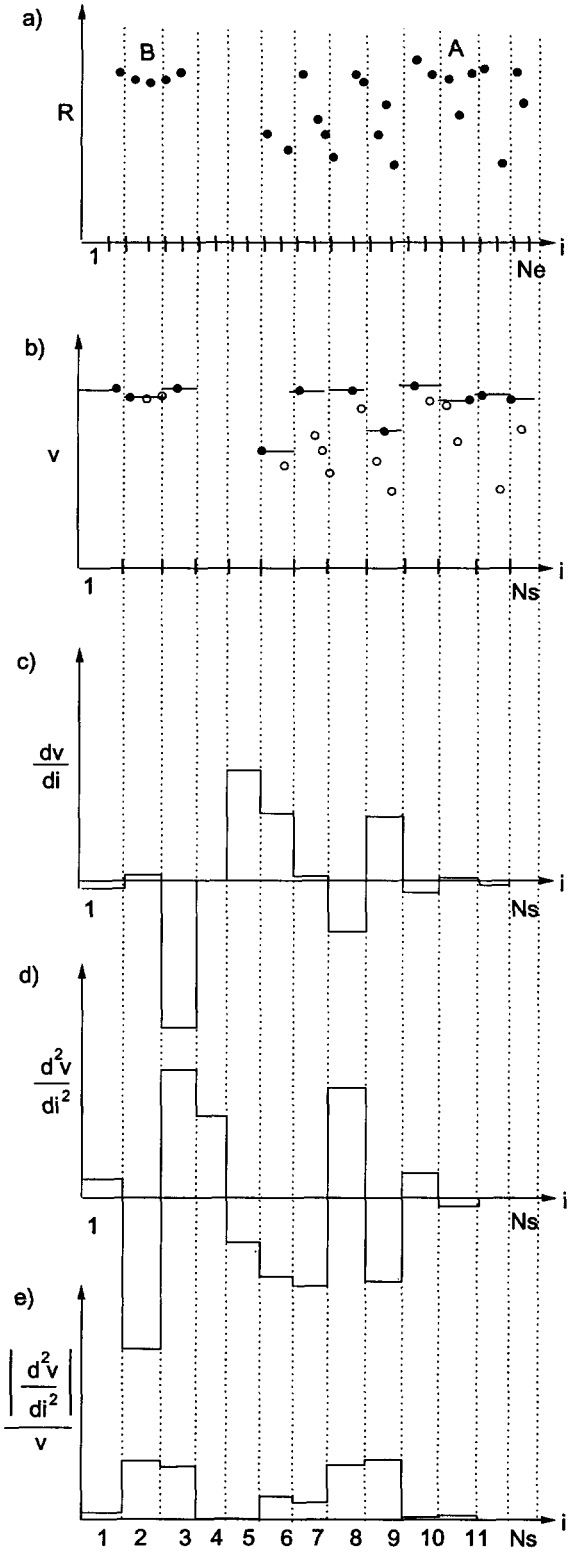

Figure 2: *a) The estimates of the example shown in Fig. 1, b) the array v of maximum range values, c) its derivative, d) second derivative and e) the absolute value of its second derivative divided by v*

tions of these derivatives for the example, found using forward differences. If the face of the obstacle, as mapped on the array v, is either constant or follows a ramp function, then the magnitude of the second derivative is small. Thus, we are able to identify objects of any shape and in any pose with respect to the sensor, as long as their surfaces are locally flat, by searching for areas of small magnitude in the second derivative.

The last step of this routine is to generate the array *nad2v* which stands for normalized absolute value of the second derivative of v. The absolute value is used to allow a magnitude-based thresholding of the second derivative of v. Furthermore, a normalization of *nad2v* by the distance to the object, removes the bias introduced by the fact that objects located at different distances have a different associated noise that affects the measure of their frequency.

The pseudo-code of the FIND-LOW-FREQ routine is

```
FIND-LOW-FREQ (A, R)
1   v(1 : N_s) ← 0
2   for r ← 1 to N_e
3      i ← ⌊A(r)/α⌋ + 1
4      if R(r) > v(i)
5         v(i) ← R(r)
6   for i ← 1 to N_s − 1
7      dv ← v(i + 1) − v(i)
8   for i ← 1 to N_s − 2
9      if v(i) = 0
10        nad2v ← 0
11     else
12        nad2v ← |dv(i + 1) − dv(i)| / v(i)
13  return (v, nad2v)
```

Lines 1-5 filter R and generate v. Lines 6-7 approximate the derivative of v using forward differences and, finally, lines 8-12 generate the absolute value of the second derivative of v normalized by v.

4.2 Selecting candidates

The GET-OBST-CANDIDATES routine selects from R those returns that are most likely to be obstacles returns. The pseudo-code of this routine is

```
GET-OBST-CANDIDATES (A, R, v, nad2v)
1   R_c(1 : N_e) ← 0
2   for i ← 1 to N_s
3      T_k(i) ← K_1 + max (T_{k−1}(i), T_{k−2}(i) )/2
4   done ← 0
5   while done = 0
6      done ← 1
7      for r ← 3 to N_e − 2
8         i ← ⌊A(r)/α⌋ + 1
9         if v(i) = 0 or R(r) = Rc(r) or R(r) = 0
10           continue
11        if (R(r) = v(i) and
           nad2v(i) = min( nad2v(i), T_k(j)|_{j=i−2}^{i+2})
12           R_c(r) ← R(r)
13           T_k(i) = T_k(i) + δ
14           T_k(i ± 1) = T_k(i ± 1) + δ/2
15           T_k(i ± 2) = T_k(i ± 2) + δ/3
16           done ← 0
17  T_{k−2} ← T_{k−1}
18  T_{k−1} ← T_k
19  return (R_c)
```

It returns an array of candidates R_c which is zero except at the locations of returns for which there is a strong evidence that belong to obstacles.

A small magnitude of $nad2v(i)$ indicates a signal with low frequency components which might indicate the presence of obstacles. For example, in the plot of *nad2v* shown in Fig. 2.e, of the five locations with a small magnitude, the locations 1 and 10 correspond to obstacles while 4 and 5 correspond to low frequency areas caused by a lack of signal. Thus, an obstacle in an area i can be observed if the value of $nad2v(i)$ is smaller than a threshold $T_k(i)$. In lines 2-3 we set this threshold to the sum of a constant K_1 and a function of the thresholds found for this location in the two previous scans. We describe the function that we used in Line 3; other functions that raise the threshold at locations where threholds were large in previous scans could also be suitable.

The main part of the routine updates the arrays of candidates R_c and thresholds T_k. A fast-rejection condition, in line 9, rejects those areas of the scan with no returns (i.e., $v(i) = 0$), individual locations with no return (i.e., $R(r) = 0$) and locations that already contain a candidate (i.e., $R(r) = Rc(r)$). The update condition, in line 11, states that a return is likely to be from an obstacle if it was selected by the maximum-value filter (i.e., R(r) = v(i)) and, if together with its four closest neighbors, it spans a low frequency region (i.e., *nad2v* is smaller than all the thresholds within a vecinity of 2). If the update condition is verified, the array of candidates is updated with the value of the estimate (i.e., $R_c(r) = R(r)$) and, having detected this position as an obstacle, all the thresholds of the neighborhood are raised by some fraction of a value δ. As shown in lines 13-15, we raise higher the thresholds of locations close to the identified obstacle than those away from it.

4.3 Reevaluating returns near to candidates

The non-zero locations of R_c are very likely to be obstacles but to achieve this confidence we sacrificed resolution that we now must recover. At this point, a value of R_c can only be a candidate if it has the

value selected by the maximum-value filter. The goal of RELAX-OBST is to use the information in R_c about the location of each obstacle to reevaluate the returns inmedeately adjacent to it. The pseudo-code of this routine is

```
RELAX-OBST (A, R, R_c)
1   done ← 0
2   while done = 0
3      done ← 1
4      for r ← 3 to N_e − 2
5         i ← ⌊A(r)/α⌋
6         C ← 1 + K_2 (C_{k−1}(i) + C_{k−2}(i))
7         T ← C (a R(r) + b)
8         if R_c(r) = 0 and
              ⋃_{j=−2}^{2} [(R_c(r + j) > 0 and
                            |R(r) − R(r + j)| < T]
9            R_c(r) ← R(r)
10           done ← 0
11  return (R_c)
```

The main loop updates the array of candidates R_c until there is a complete reevaluation of the returns of R that does not lead to an update. As indicated in lines 8-9, a return is updated to be a candidate if it was not previously a candidate and if any of its adjacent returns is both already a candidate and is such that the difference between the two returns is smaller than a threshold T. Hence, we are exploiting the spatial locality of the obstacles to reevaluate a return adjacent to a return that is known to belong to an obstacle; if two adjacent returns hit targets located at about the same distance and if we know that one of the targets was an obstacle then, it is very likely that the other return hit the obstacle too.

The key for a succesful updating is the threshold T which is composed of two elements, as described in lines 6-7. The first element is the scalar C, larger than unity, which is some function of the arrays C_{k-1} and C_{k-2} that contain information about the location of obstacles in the previous two scans, i.e., these arrays keep track of obstacle likelyhood over time in the same way that the arrays T_{k-1} and T_{k-2} kept track of previous thresholdings in the GET-OBST-CANDIDATES routine. We will describe these arrays later, when we discuss the routine where they are updated. The second element of T is our approximation to the error of the estimate, i.e., a linear function that increases with range with parameters given by Eq. 1. Thus, a return adjacent to a candidate return is updated to be a candidate if the difference between their ranges lies within the expected error for that distance; if we have evidence that an obstacle has been sighted before at this location, then $C > 1$ relaxing the expected error.

This concludes the selection of returns that are likely to belong to be returns from an obstacle.

5 Candidate prunning

The second three routines of CLASSIFY return a final classification R_f of each return based on prunning the array of candiates R_c. The objective of this prunning is to keep a low false alarm rate.

The first prunning exploits the time and space locality of the obstacles. The code of this routine is

```
REMOVE-NEW-OBST (A, R_c)
1   for r ← 3 to N_e − 2
2      i ← ⌊A(r)/α⌋ + 1
3      if ⋂_{j=i−2}^{i=2} (C_{k−1}(j) = 0 and C_{k−2}(j) = 0)
4         R_f(r) ← 0
5      else
6         R_f(r) ← R_c(r)
7   return (R_f)
```

Lines 4-7 state that a candidate return $R_c(r)$ must be added as a final obstacle $R_f(r)$ only if an obstacle was located in the neightborhood in either of the last two scans. An obstacle at a given location will be identified as a previously seen obstacle regardless of whether it moves slightly or is temporarily occluded. Clearly, the obstacle will be identified as a new obstacle (and not added to R_f) if its position with respect to the ladar changes so fast that its new location is outside its previous neightborhood or if the occlusion lasts so long that the evidence of having seen it before has expired.

The second prunning routine, REMOVE-THIN-OBST, removes from the list of final obstacles those which appear to be isolated hits, i.e, We keep an obstacle only if there are O_s obstacles within a surrounding window of $\pm W_s/2$ returns where $W_s \ll N_e$. The code of the routine is arby.

```
REMOVE-THIN-OBST (R_f, W_s, O_s)
1   d = ⌊W_s/2⌋
2   T[1 : N_e] ← R_f[1 : Ne]
3   for r ← d + 1 to N_e − d
4      if T[r] > 0
5         cont ← 0
6         for k ← −d to d
7            if T[r + k] > 0
8               cont ← cont + 1
9         if cont < O_s
10           R_f[r] ← 0
11  return (R_f)
```

The last pruning routine is CROP-RANGE which

crops the array of obstacles R_f to the range for which our linear approximation to the expected error of an estimate holds. The routine is

```
CROP-RANGE (R_f, d_min, d_max)
1  for r ← 1 to N_e
2     if R_f(r) > d_max or R_f(r) < d_min
3        R_f(r) ← 0
4  return (R_f)
```

The complexity of each one of these prunning routines is $O(N_e)$.

6 Saving the state for the next scan

The last routine of CLASSIFY updates the arrays C_{k-1} and C_{k-2} which keep track of the obstacles over time and returns the array R_s that summarizes the position of the obstacles and the flag *alarm*. The code of the routine is

```
UPDATE-AND-ARCHIVE (A, R_c)
1  C_k(1 : N_s) ← 0
2  R_s(1 : N_s) ← 0
3  alarm ← 0
4  for r ← 1 to N_e
5     i ← ⌊A(r)/α⌋ + 1
6     if R_c(r) > 0
7        C_k(i) ← K_3
8        if R_s(i) = 0 or R_s(i) < R_c(r)
9           R_s(i) ← R_c(r)
10       alarm ← 1
11 C_{k-2} ← C_{k-1}
12 for i ← 1 to N_s
13    if C_k(i) > C_{k-1}(i)
14       C_{k-1}(i) ← C_k(i) + C_{k-1}(i)
15    else
16       C_{k-1}(i) ← C_{k-1}(i) − ⌈(C_k(i) + C_{k-1}(i))/2⌉
17       if C_{k-1} < 0
18          C_{k-1} ← 0
19 return (alarm, R_s)
```

The first loop of the routine, in lines 4-10, accomplishes three things. First, it initializes the array C_k with a value K_3 in all the locations where a candidate was found. We use the array of candidates R_c to initialize C_k instead of the array of final obstacles R_s so we can reduce the alarm rate (generating R_f) without losing information about the previous appearance of the candidates (contained in R_c). The value of K_3 is selected to be of the order of magnitude of the expected error of the estimates. Second, in lines 8-9, it sets the value of R_s to the closest non-zero return, i.e., if there is an obstacle in a given direction i, then $R_s(i)$ will contain the minimum distance between the robot and the obstacle. This value can be used to modify a robot trajectory without having to analyze the individual returns. Finally, the first loop raises the *alarm* flag after it finds that there is, indeed, an obstacle in the scan.

Lines 11-18 update the arrays that have evidence of the presence of obstacles. Line 11 copies C_{k-1} into C_{k-2}. The loop in lines 12-18 update the vector C_{k-1} in the following way. If there is a stronger evidence of the presence of an obstacle than the evidence tha we had before (i.e., $C_k(i) > C_k(i+1)$) then bias the evidence in a positive direction; otherwise, bias the evidence the the negative direction. In our case, we have bias the evidence in the positive direction by adding the evidence of C_k and C_{k-1}. This is a strong bias that makes the algorithm aware of the presence of an obstacle very fast. In contrast, we biased the evidence in the negative direction by substracting from C_{k-1} the average of the evidence of C_k and C_{k-1}. This is a weak bias that makes the algorithm forget the presence of an obstacle very slow. Thus, obstacles that have been seen before, even before many scans, are rapidly identified when they reappear. Lines 17-18 make sure that the evidence is never negative.

7 Experimental results

Although the algorithm has a number of parameters that appear that are loose, in practice, finding the appropriate values is easy. These type of parameters appear often in systems that deal with real world uncertainties and there is no other way to adjust them but a trial-an-error approach. To put the issue in perspective, consider that a control system based on a PID compensator also requires this type of adjusting but still, it is the most commonly used compensator. Likewise, there is no way to determine, for example, which is the best size of a secondary cache memory given the size of the primary cache, the RAM and the hard disk. A given configuration that excels in a given benchmark will lag in another. Still, it is very easy to estimate the size of a secondary cache that is likely to work well under a large number of computational loads. In summary, the parameters of the routines that we presented are quite easy to adjust and once set, the algorithm works well in a wide variety of scenarios.

The algorithm was designed using the same set of data in related work by Macedo, Manduchi and Matthies [5]. The data consists of eight sequences with a lenght that varies between 15 and 90 seconds. The platform used to gather the data was the Urbie robot using and Accuity AccuRange 4000 ladar. This ladar operates at a wavelength of 1064 nm and has sampling interval of $\theta = 12.2$ mradians. Each scan

has $N_e = 512$ returns and thus, it covers an angle of $\Omega = 2\phi$. The laser is capable of estimating range up to distances of 50 ft. Still, due to the size and speed of the robot and we kept our range of interest within a minimum distance $d_{min} = 0.3$ m and a maximum distance of $d_{max} = 2$ m. Given these parameters, the laser can hit a 2.5 cm diameter pole 2 times at a distance of 1 m. We measured the error of the estimates for two known targets within this range and obtained that the coefficients in Eq. 1 are $a = 0.015$ and $b = 12.3$. This means that we expect the error to rise at a rate of 1mm evey 70mm of range. The free parameters in GET-OBS-CANDIDATES were set to $K_1 = 0.025$ and $\delta = 0.03$. The free parameter in RELAX-OBST was set to $K_2 = 0.05$. Finally, the free parameter in UPDATE-AND-ARCHIVE was set to $K_3 = 25$.

Once the parameters of the algorithm were set, they kept fix for all the scenenarios of the data set. The results for this set is 4 false alarms in 800 images which covered scenerios where the robot was in the clear, in sparse grass and in dense grass, both while standing still and moving, both with and without rocks clear and hidden by grass. Figure 3 shows a robot approaching two rocks. The corresponding graph of returns, as seen by the laser, is shown in Fig. 4. The graphs shows an area of $2 \times 2/, m^2$. The front of the robot is upwards. The laser is located at the origin of the graph. The solid pie slices originating from it are robot self-occlusions caused by the on-board camera, the antennas and other structures. The long lines originating from the laser indicate returns that were identified as NOTFOLIAGE. All other returns (i.e., the dots of the graphs) were identified as foliage.

In Fig. 5 and 6 we have the corresponding scene and graph of the robot travelling near a field of very dense grass. Notice that in front of the robot there is a solid wall of grass which the algorithm correctly identifies as foliage. The robot is partially tilted to the right and the scan hits the ground. The corresponding returns are also correctly identified as members of the NOTFOLIAGE class.

The algorithm was ported to Urbie which, by the time, had a different ladar system. The new ladar system is a Sick laser with a sampling interval of $\theta = 8.7$ mradians (i.e., 0.5 degrees), covers an angle of $\Omega = \phi$ in each scan and has an expected error in the range of 70 mm at 4 m. The adjustment of the parameters was easy and the robot perform correcly at the first try, during the first field test. The algorithm runs in real time and drains minimal resources from the robot.

Figure 3: Robot facing two rocks at 45° and 315°

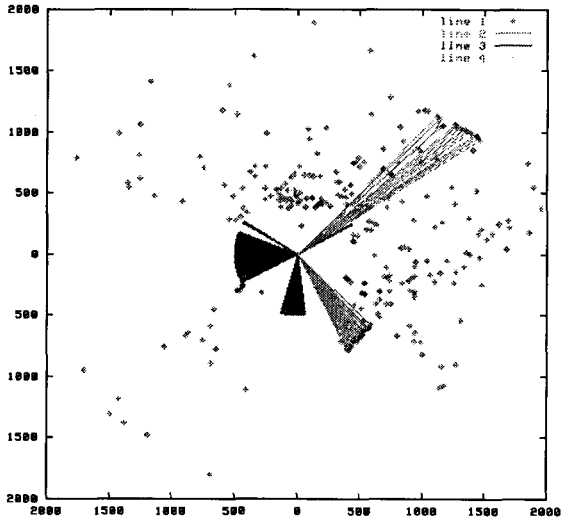

Figure 4: Return classification of scene in Fig. 3

8 Conclusion

We have presented a real time algorithm to identify foliage present in a natural scene. The algorithm uses the locality of the obstacles in time and space to first identify a list of returns that were likely to hit obstacles and then to prune this list to eliminate false positives. The algorithm was succesfully ported to the Urbie robot. Our future plans are to extend this algorithm to handle range images.

Acknowledgments

The collection of the data set, the development of the algorithm and its implementation in the Urbie robot were sponsored by the DARPA MARS program under contract ******* and the DARPA TMR program under contract ********.

References

[1] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer, "Vision and navigation for the carnegie-mellon navlab,"

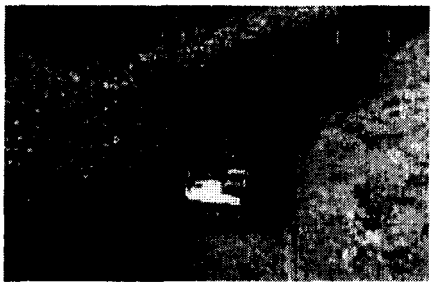

Figure 5: *Robot with field of grass on the left*

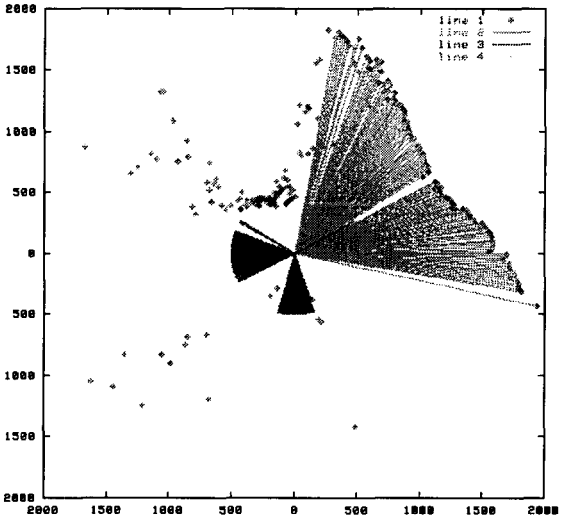

Figure 6: *Return classification of scene in Fig. 5*

IEEE Trans. Pattern Anal. Machine Intell., vol. 10, pp. 362–373, May 1988.

[2] M. Adams, "Lidar design, use, and calibration concepts for correct environmental detection," *IEEE Trans. Robotics Automat.*, vol. 16, pp. 753–761, Dec. 2000.

[3] M. Hebert and E. Krotkov, "3-d measurements from imaging laser radars: how good are they?," *Intl. Journal Image and Vision Computing*, vol. 10, pp. 170–178, Apr. 1992.

[4] I. S. Kweon, R. Hoffman, and E. Krotkov, "Experimental characterization of the perceptron laser rangefinder," Tech. Rep. CMU-RI-TR-91-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Jan. 1991.

[5] J. Macedo, R. Manduchi, and L. Matthies, "Ladar-based discrimination of grass from obstacles for autonomous navigation," in *Proc. Intl. Symposium on Experimental Robotics*, pp. 111–120, 2000.